1.3 The Chess Machine: An Example of Dealing with a Complex Task by Adaption

Allan Newell

The modern general-purpose computer can be characterized as the embodiment of a three-point philosophy: (1) There shall exist a way of computing anything computable; (2) The computer shall be so fast that it does not matter how complicated the way is; and (3) Man shall be so intelligent that he will be able to discern the way and instruct the computer.

Sufficient experience with the large machine has accumulated to reveal the peculiar difficulties associated with these points. There has been a growing concern over problems which violate them, and instead satisfy the condition that (1) The relevant information is inexhaustible; (2) The set of potential solutions is neither enumerable nor simply representable; (3) The processing to be accomplished is unknown until other processing is done; and (4) An acceptable solution is required within a limited time.

Most design problems, including programming a computer, are of this nature; so are the very complex information processing tasks like translating languages or abstracting scientific articles. The current arguments about thinking machines and general-purpose robots also revolve about whether computers can deal with problems of this general nature.

The problem of playing good chess certainly falls into this class of ultracomplexed problems. It is a useful type case for general discussion because the nature of the task and the complexities surrounding it are common knowledge. Further, it already has something of a history. (Shannon, 1950; Weinberg, 1951; Richards, 1952; Strachey, 1952)

The aim of this effort, then, is to program a current computer to learn to play good chess. This is the means to understanding more about the kinds of computers, mechanisms, and programs that are necessary to handle ultracomplexed problems. The limitation to current computers provides the constant reminder that the heart of the problem lies in the limitation of resources, both memory and time. This aim would be somewhat ambitious even if all the details were available. The paper will actually be limited to presenting an over-all schema which appears feasible, and relating it to some of the critical problems which must be solved. The reference to learning expresses the conviction that the only way a machine will play good chess is to learn how. Although learning considerations have been prominent in the thinking and motivation behind the machine, attention will have to be restricted to the performance system; that is, to those features which are necessary in order to play the game. However, some of the learning potentialities implicit in the performance system will be discussed.

The work presented here represents the early phases of an attempt to actually construct such a program for the JOHNNIAC, one of Rand's high speed computers.

Before starting it is desirable to give some additional conditions of the problem. From now on the computer with program will be called the machine, and the various parts of what it does will be called mechanisms. The problem is not to construct a machine which can induct the rules of chess by playing; it will be instructed concerning the legalities. Finally, the machine is only to do the job of one man; it will require an outside opponent, human or otherwise.

Problems

As everyone knows (von Neumann et al., 1947), it is possible in principle to determine the optimal action for a given chess position. First compute out all...
continuations to the bitter end. See whether they win, lose, or draw; and then work backwards on the assumption that the opponent will always do what is best for him and you will do what is best for you.

The difficulty, of course, is that this in principle solution requires a rather unnerving amount of computing power, and doesn't give any clues about what to do if you don't have it. It will provide us, however, with a short checklist of problems which must be solved if the computing requirements are ever to shrink to a reasonable size.

The most striking feature of the in principle solution is the tremendous number of continuations. This is accounted for by both the number of new consequences that appear at each additional move into the future, and the large number of moves required to explore to the end. This provides two problems:

1. The consequences problem, or which of the possibilities that follow from a given proposed action should be examined;
2. The horizon problem, or how far ahead to explore.

The possibility that one might stop looking at some intermediate position only raises a third problem:

3. The evaluation problem, or how to recognize a good position when you see one.

Another feature of the in principle solution is the identical examination of all the possible alternative actions. Despite the similarity in describing both present and future moves, it is worth while to keep distinct the actions that are actually available at a move, and from which a choice must be made; and the future consequences of these actions, which may include, among other things, limitations on the alternatives available in the future. Hence, we have;

4. The alternatives problem, or which actions are worth considering.

These four problems, consequences, horizon, evaluation, and alternatives, will be sufficient to keep us aware of the difficulties as we search for a set of mechanisms to play chess. Solutions must be found to all of them if the machine is to play good chess with reasonable resources.

Overview

There is a common pattern to the solutions to be described here. In all of them the machine uses very partial and approximate methods. It is as if the machine consisted of a vast collection of rules of thumb. Each rule is a much oversimplified expression of how the machine should behave with respect to some particular aspect of the problem. The rules are of all kinds: chess principles to follow, measurements to make, what to do next, how to interpret rules of thumb, and so on. These are so organized that they form the necessary qualifications and additional specifications for each other. Each rule is essentially a machine program. At any particular instant the machine is under the control of some such rule, or shunting between rules under the control of a master program.

The main effort of the paper is devoted to describing how such a set of rules can be defined and organized to achieve solutions to the four problems, and thus provide a schema for a machine which puts all these pieces together to play chess. Only minor effort is devoted to indicating the detailed structure of these programs at the level of machine code.

One aspect of the underlying coding does require attention, and is dealt with at the end of the paper. The large number of rules, their complexity, and the necessity for adding new ones and modifying old ones, implies the use of a fairly extensive general-purpose language. That is, all these rules are to be given in this language or pseudo-code, as it might also be called. Hence, each use of a rule must be preceded by an interpretive step. However, a few programs suffice for using any and all of the rules that might be required in the machine.

Preliminaries

Let us start by providing the machine with a few basic facilities. Each chessman and each square of the chessboard needs a name, suitably coded into binary bits. A fixed set of addresses are set aside to hold the current position. This can be given as a list of the men with the squares they occupy, including a "zero" square if the man is off the board. The machine can accept an opponent's action by reading a punched card. This can be given as a list of the men which have been moved, along with their new locations. Thus an action